Order Management

Bardi Bogdan
Group:30421
Year of Study: 2019-2020

Contents

1 Requirements

The main goal of this assignment was to design and implement an application for processing customer orders for a warehouse.

To achieve the main objective I had to go through many different steps which will be thoroughly explained in the following chapters:

1. Analyzing use cases and the problem itself
2. Creating an UML Diagram and designing the necessary classes and data structures
3. Designing the user interface
4. Implementing the created designs
5. Creating unit tests

2 Problem Analysis

In order to manage orders for a warehouse we need a database to store the necessary data to do so such as customers' name and address, product on stock, and the history of previous orders. To do so we use a relational database management system(RDBMS) such as MySQL or Microsoft SQL.

This application will process commands from a text file and will perform CRUD(Create,Read,Update,Delete) operations to the database and to be able to create Reports and Invoices when asked. The operations needed to be implemented are as follows:

- Add clients to the database
- Delete clients from the database
- Add products to the database
- Delete products from the database
- Create order for client
- Generate Reports

The user will create a text file which contains the commands for the program to be executed and then will pass that text file as an argument to the software. Essentially the application provides an abstraction layer between the user and the database which simplifies usability and not requiring the user to learn SQL in order to do queries to the database.

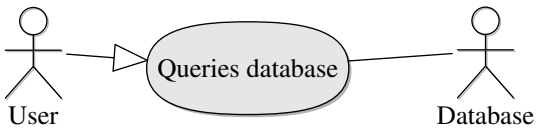

Figure 1: Use case diagram

3 Program Design

3.1 Database Design

In order to use the relational database more effectively each information is split into individual tables which are then linked at query time by their keys in order to avoid redundancies as much as possible(as an example instead of storing addresses/cities into the clients table we can separate those as more than 1 client can stay at the same address).

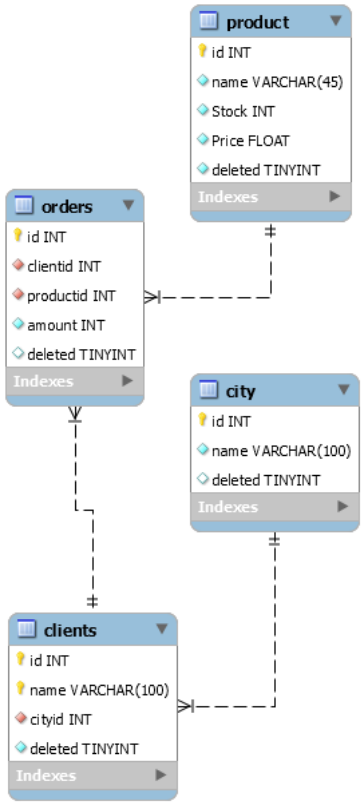

Figure 2: Database Diagram for the Warehouse

3.2 Command Syntax

In order to process the commands from the text file a parser has to be implemented in order to process the commands and those commands have to follow a specific syntax in order to be recognized correctly.

3.2.1 Add client

Syntax: Insert client: Ion Popescu, Bucuresti
Description: Inserts into the database a new client with name Ion Popescu and address Bucuresti

3.2.2 Add product

Syntax: Insert product: apple, 20, 1
Description: Inserts into the database a new product called apple with quantity 20 and price 1

3.2.3 Delete product

Syntax: Delete product: apple
Description: Deletes the product apple from the database

3.2.4 Delete client

Syntax: Delete client: Ion Popescui
Description: Deletes the client with name Ion Popescu from the database.

3.2.5 Create order for client

Syntax: Order: Ion Popescu, apple, 5 Description: Creates a new order for Ion Popescu with apple quantity 5. It decrements the apple stock by 5 and generates a bill for it.Each order command represents a new order

3.2.6 Generate Reports

Syntax: Report client/order/product Description: Generates a new report with all clients/orders/products.

3.3 Class Design

The class structure of this program follows a layered architecture which consists of splitting the application into different layers, each with its own specific purpose and calls functions from below it

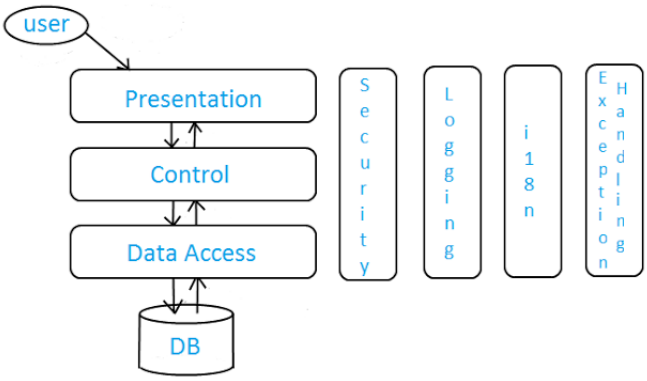

Figure 3: Example of a layered architecture

UML, short for Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems. It allows us to visualize the class structure of the three aforementioned packages and how they interact with each other.

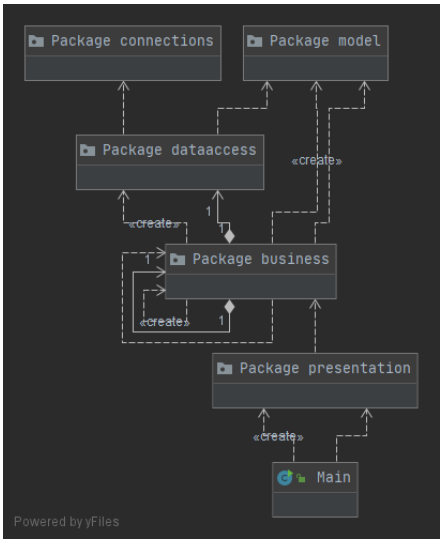

Figure 4: Package structure of my application

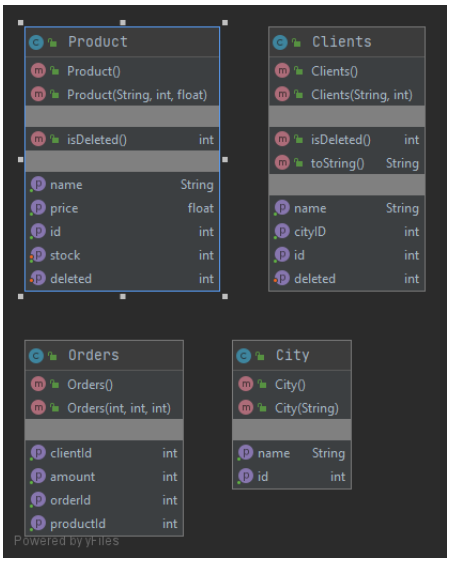

Figure 5: Model package

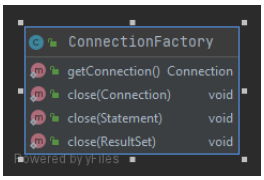

Figure 6: Connections package

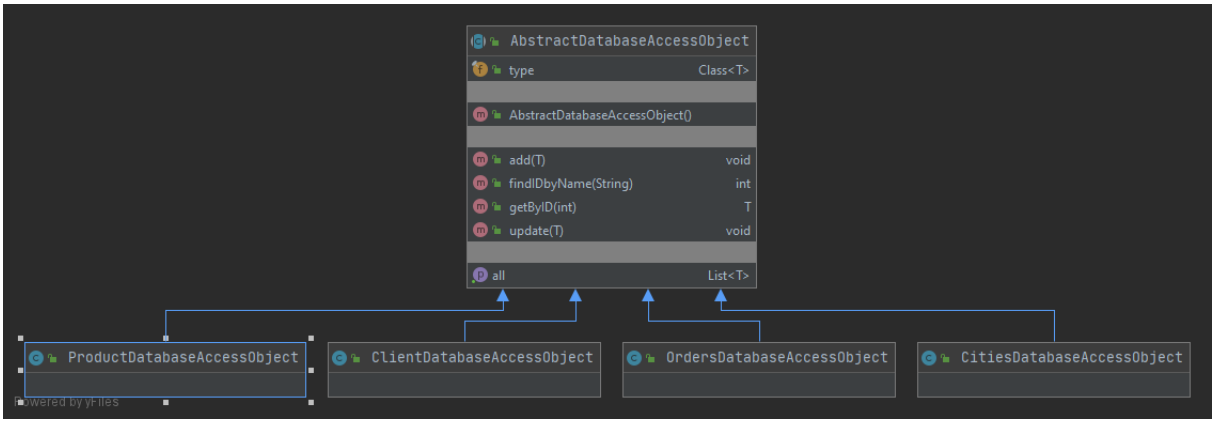

Figure 7: Data access package

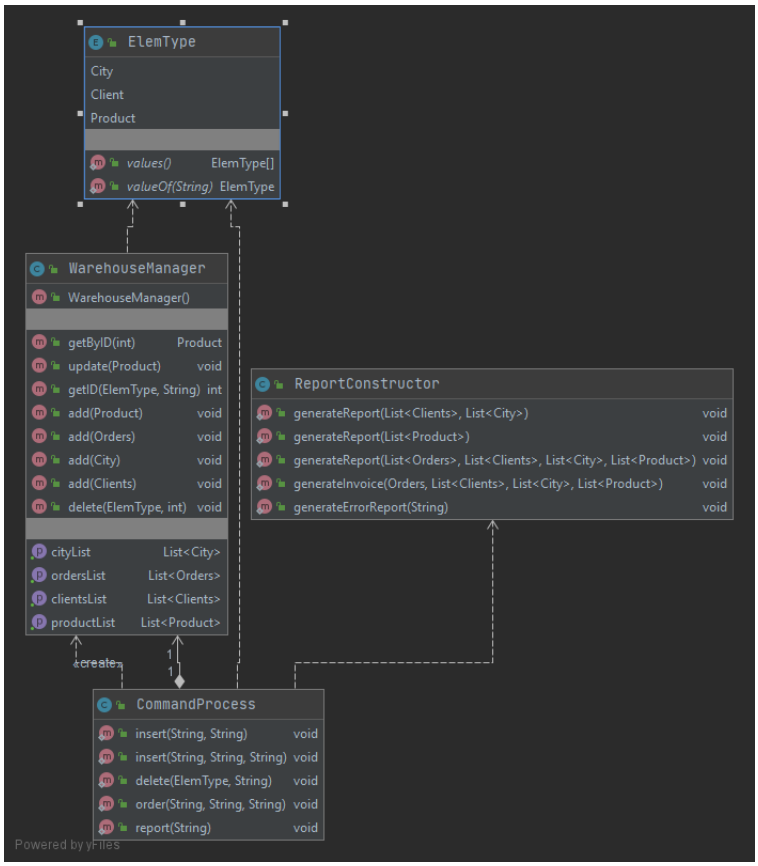

Figure 8: Business package

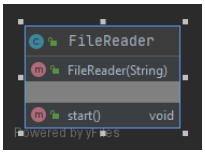

Figure 9: Presentation package

4 Implementation

This chapter will describe all classes which make up the application.

4.1 Connections

The Connections package provides the singleton class ConnectionFactory which provides static methods to create and close connections between the application and the DBMS.

4.1.1 getConnection

Creates a new connection to the database or fetches an existing one and returns it.

4.1.2 close(Connection/ResultSet/Statement)

Overloaded method which closes the Connection/ResultSet/Statement given in the argument.

4.2 AbstractDatabaseAccessObject

This is an abstract class which provides the genertic CRUD operations which are generated using generic classes and reflection techniques in order to select the proper table and get the necessary fields. This class' methods will be inherited by the other classes in the dataaccess package.

4.2.1 add(T)

Method to add the generic model paramenter to its corresponding table in the database using reflection methods. It creates the Insertion query inserts the data necessary to complete it such as the name of the table,fields to be filled and their corresponding data.

4.2.2 findIDbyName(String name)

Method to find an entry in the database based on its name, it will not work with all tables but its useful for finding clients or products based on their names. It returns their id in the database.

4.2.3 getByID(int id)

Fetches the model from the database based on its id. It returns null if it doesn't find it.

4.2.4 getALL()

Returns a list of all elements in one table, the table is dependent on the class which inherited this method

4.2.5 update(T)

Updates an entry in the database based on the id inside the generic T element.

4.3 ClientDatabaseAccessObject

This class provides the necessary methods to process all database access regarding the client table. It gets its methods from the abstract class which has the generic parameter a Client class.

4.4 CitiesDatabaseAccessObject

This class provides the necessary methods to process all database access regarding the Cities table. It inherits those methods from the abstract class which has the generic parameter a Cities class.

4.5 ProductDatabaseAccessObject

This class provides the necessary methods to process all database access regarding the product table. It uses the methods implemented in the abstract class but with a Product class as the generic parameter.

4.6 OrdersDatabaseAccessObject

This class provides the necessary methods to process all database access regarding the orders table. Like all classes in the dataaccess package it inherits its methods from the abstract class AbstractDatabaseAccessObject and has the Order class as its generic parameter.

4.7 City

It provides the necessary data structure and methods to represent an entry in the city table

4.7.1 Important fields

- id - id inside the table
- Name - name of the city

4.7.2 getName()

Gets the name of the city.

4.7.3 getID()

Fetches the id of the city.

4.8 Clients

It provides the necessary data structure and methods to represent an entry in the clients table

4.8.1 Important fields

- id - id inside the table
- Name - name of the client
- cityID - city identifier inside the table
- deleted - flag to check if the client was previously deleted or not

4.8.2 getName()

Gets the name of the client.

4.8.3 getID()

Fetches the id of the client.

4.8.4 getCityID()

Fetches the id of the city the client resides in.

4.8.5 isDeleted()

Checks if the client was deleted or not.

4.8.6 setDeleted()

Sets the deleted flag to the value in the parameter.

4.9 Product

It provides the necessary data structure and methods to represent an entry in the products table

4.9.1 Important fields

- id - id inside the table
- Name - name of the product
- stock - amount of product currently in stock
- price - price of the product
- deleted - flag to check if the client was previously deleted or not

4.9.2 getName()

Gets the name of the product.

4.9.3 getID()

Fetches the id of the product.

4.9.4 setStock()

Updates the stock of the product.

4.9.5 getStock()

Gets the amount of product currently in stock.

4.9.6 getPrice()

Fetches the price of the product.

4.9.7 isDeleted()

Checks if the client was deleted or not.

4.9.8 setDeleted()

Sets the deleted flag to the value in the parameter.

4.10 Orders

It provides the necessary data structure and methods to represent an entry in the orders table

4.10.1 Important fields

- id - id inside the table
- clientId - id of the client who placed the order
- productId - id of the product ordered
- amount - amount of product ordered

4.10.2 getOrderID()

Gets the id of the order

4.10.3 getClientID()

Fetches the id of the client who ordered the product.

4.10.4 getProductID()

Fetches the id of the ordered product.

4.10.5 getAmount()

Fetches the id of the amount ordered.

4.11 CommandProcess

Wrapper to process the commands further in order to pass them to the Warehouse Manger and actually add them to the DB.

4.11.1 insert()

Inserts a new client or product depending on the number of arguments. It translates them into their corresponding Model object and sends it to the WarehouseManager instance.

4.11.2 delete()

Deletes an entry from the database based on the type and name of the entity deleted. It finds the id and passes a delete command

4.11.3 order()

Creates a new order based on the name of client,name of product,amount of product. Queries the product and client tables to get their ids, checks the available stock and adds it if there is enough it adds it to the database and generates an invoice or creates an error report.

4.11.4 report()

Determines which kind of report to create and passes it to the ReportConstructor class.

4.12 ReportConstructor

Provides static methods to create reports and invoices pdf files using the iText library.

4.12.1 generateErrorReport(String message)

Creates an error report with the specified error message.

4.12.2 generateInvoice()

Creates a new invoice based of the order object passed as an argument.

4.12.3 generateReport()

Overloaded method which generates a new report based on the number of lists given(1 for Product, 2 for clients,4 for orders).

4.13 WarehouseManager

Class which composes all DAOs and provides methods to interact with them and perform the CRUD operations.

4.14 ElemType

Enum needed to decode which kind of element(City,Client or Product) is needed to be fetched. Used by WarehoudeManager and CommandProcess.

4.15 FileReader

It opens a file scanner and reads line by line and processes the commands like this:

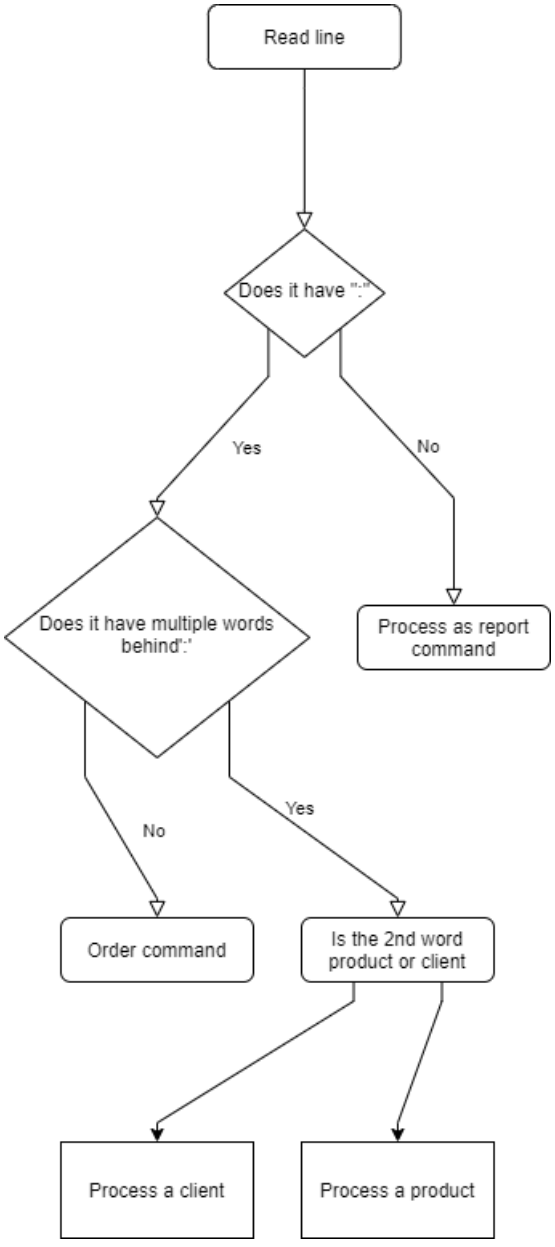

Figure 10: Decision tree of the FileReader

5 Testing

There was a basic command list that had to be run and provide its output inside the repository. The file basically tests if the elements from the database are inserted and removed properly, that error conditions that may arise from the order command(such as out of stock errors) are properly handled.

```
Insert client: Ion Popescu, Bucuresti
Insert client: Luca George, Bucuresti
Report client
Insert client: Sandu Vasile, Cluj-Napoca
Report client
Delete client: Ion Popescu, Bucuresti
Report client
Insert product: apple, 20, 1
Insert product: peach, 50, 2
Insert product: apple, 20, 1
Report product
Delete Product: peach
Insert product: orange, 40, 1.5
Insert product: lemon, 70, 2
Report product
Order: Luca George, apple, 5
Order: Luca George, lemon, 5
Order: Sandu Vasile, apple, 100
Report client
Report order
Report product
```

6 Conclusions

This assignment is a great usage of the Layered architecture in Java OOP design and it highly encourages the programmer to use Reflection techniques to optimize the design and code bloat while also introducing the programmer to libraries such as iText and the MySQL connector.

7 Bibliography

- http://www.mkyong.com/jdbc/how-to-connect-to-mysql-with-jdbc-driver-java/
- http://theopentutorials.com/tutorials/java/jdbc/jdbc-mysql-create-database-example
- https://dzone.com/articles/layers-standard-enterprise
- http://tutorials.jenkov.com/java-reflection/index.htm
- https://www.baeldung.com/java-pdf-creation
- https://www.baeldung.com/javadoc